Gaps in Software Testing Education: A Survey of Academic Courses in Sweden

Ayodele A. Barrett
Mälardalen University
Västerås, Sweden
ayodele.barrett@mdu.se

Eduard P. Enoiu
Mälardalen University
Västerås, Sweden
eduard.paul.enoiu@mdu.se

Wasif Afzal
Mälardalen University
Västerås, Sweden
wasif.afzal@mdu.se

***Abstract*—A cross-sectional, questionnaire-based survey of software testing courses offered at Swedish universities was undertaken in the final quarter of 2023. With a return rate of 44%, the survey delved into the contents of these software testing courses to gain an understanding of how the courses differ in terms of depth and breadth of content. Information was also sought about administrative and course planning activities related to the courses. Some key findings are that there is in-depth coverage of unit testing in all the courses, with none of the courses offering in-depth testing of other test levels such as acceptance testing. Also notable is the difference in test types. As an example, functional testing is taught in-depth in all the courses, while accessibility testing is not taught at all in half of the courses. It is suggested that a greater range of software testing topics is needed in future education if more stakeholders, such as business analysts and software developers, not just software testers, are to have a quality-centred approach to software development.**

***Index Terms*—survey, academic software testing education, curricula, software testing, higher education, software quality, Sweden**

I. Introduction

Software testing is carried out to demonstrate that software is fit for purpose and to detect defects within the software. Ensuring the quality of software should not be seen only as the responsibility of software testers. Other stakeholders in the development process, such as developers and business analysts, will also benefit from learning how to conduct software testing.

In research that focuses specifically on software testing education, a general consensus that has spanned the decades is that too few software testing courses have been, and continue to be, offered at universities [3, 27, 33]. There is also acceptance that existing computing (Computer Science, Information Systems, Information Technology and related) curricula have already demanding schedules, with some curricula immutable for a number of years. The often-limited availability within curricula leaves little room for the introduction of additional courses. In particular, computing degrees are disciplines with ever-expanding topics. Thus, the development of additional courses, specifically to address the dearth of software testing courses, is understandably unrealistic in the short term [11].

Software testing is taught either in dedicated courses or embedded in other classes, such as general software engineering or programming courses. With the latter, it has been suggested that due to an often-packed syllabus and a lack of time, course content related to software testing is not given enough emphasis [11]. This situation is not unlike that in industry where software testing activities are similarly curtailed when there are time constraints [24]. Hence, it would appear that in the industry as it is in academia, testing is a topic that tends to be pushed to the side [16, 19]. It is therefore of great importance to ensure that available software testing courses, although few, are thorough and varied enough to inculcate students with a quality-oriented mindset, which could lead to more reliable code [17].

With data collected from software testing lecturers in Swedish universities by way of an online questionnaire, the following research questions (RQs) will be answered:

RQ1. How are software testing courses in Sweden typically structured?
RQ2. Which aspects form the basis of the course contents?
RQ3. Which software testing topics are most commonly taught in terms of breadth and depth?

By addressing these research questions it is hoped that the data and analysis from this survey could be used in updating or developing software testing courses. This is particularly important in the rapidly changing field of computing.

The paper is structured as follows: In Section II, related studies are detailed. Section III is a description of the thought process that went into conducting the survey. Section IV contains the findings of the survey results. Discussion of the results is given in Section V. Limitations of the study are presented in Section VI, and the conclusions of the paper are in Section VII.

II. Related Studies

Recent work addresses the state of software testing education in various regions of the world. The first by Barrett et al. [3] sought to identify dedicated software testing courses taught in Swedish universities that offer Computer Science or related degrees. A key finding was that 56% of universities offered at least one dedicated software testing course. Additionally, the authors found that software testing accounted for between 5% and 8% of a typical Computer Science syllabus.

Similarly, Ardic and Zaidman [2] focused on identifying universities that offer dedicated software testing courses. Their study, however, was based on a global ranking list of the top 100 Computer Science programs. As a result, the study spanned 21 countries across four continents. The authors found that 49% of the Computer Science curricula reviewed contained a course dedicated to software testing. The authors also noted a regional variance with European universities, at 61%, more likely to offer a dedicated course and Asian universities, at 25%, less likely to offer a dedicated software testing course.

Also sourced from a university ranking site, albeit one that differs from the preceding study, is the work by Tramontana et al. [29], which focuses on software testing courses offered at Belgian, Italian, Portuguese and Spanish universities. An initial list of 171 universities was whittled down to 49 for analysis. The authors found that 22 of the 49 universities considered (approximately 45%) offered dedicated software testing courses, of which the majority of the courses are being taught at the Masters level.

From a student's perspective, Cerón et al. [4] surveyed a sample of Computer Science students registered at different Ecuadorian Universities. Students were queried about their knowledge of software testing, how much of the knowledge was imparted by their lecturers, and how motivated they were to test while developing school projects. The authors inferred that while theoretical aspects of software testing were firmly rooted in Ecuadorian universities, the practical aspects were far less firmly entrenched.

Melo et al. [21] administered a survey to professors across the globe who were involved with teaching courses which addressed software testing. With responses from four continents, some findings include the fact that unit testing was the most frequently taught test level. In addition, the authors found that the most commonly taught test design techniques were equivalence partitioning and boundary value analysis.

Working from the premise that it is infeasible to have a perfect, all-round tester for the testing of large-scale software systems, Mårtensson and Sandahl [20] sought to propose a model consisting of different testing profiles that could be tailored for different types of test activities. The model was created after interviews and focus group discussions with test practitioners from companies dealing with large systems. To buttress the findings from the interviews with practitioners, the authors interviewed lecturers from six Swedish universities about their Masters-level software testing courses.

Save for the research by Melo et al. [21] and Mårtensson and Sandahl [20], the related studies discussed did not take into consideration the opinions of software testing lecturers. Cerón et al. [4] based their study on student perspectives while the studies by Barrett et al. [3], Ardic and Zaidman [2] and Tramontana et al. [29] garnered much of the information used in their studies from the websites of relevant universities. Melo et al. [21] did not focus solely on dedicated software testing courses but also included in their survey responses from lecturers whose course names included, for example, database management and network engineering. Finally, Mårtensson and Sandahl [20] focused their interviews of university professors on Master-level courses. Thus, this study seeks to explore a lecturer's perspective of the contents of dedicated software testing courses without considering which level of study the course offers.

III. Methodology

The survey was a one-off, cross-sectional survey conducted from October to December 2023. The survey was designed based on guidelines for conducting software engineering surveys provided by Kitchenham and Pfleeger [12] and Linåker et al. [18].

A. Respondent Group

A list of software testing courses was identified from the website run by the University and Higher Education Council (UHR) of Sweden[1]. There were an initial 26 courses identified offered by 16 universities. Each entry has a hyperlink to the university offering the course. Every course website had contact details for the lecturer responsible for the course or email addresses for course administrators. For the latter, an email was sent initially to request the responsible lecturer's contact details.

Further examination of the list of courses revealed that the two courses were the same, offered in two different universities but taught by the same lecturer. Another two courses were found to be the same, provided at the same university, taught by the same lecturer but at different academic levels - one at the undergraduate level and the other at the Masters level. Additionally, two other courses had been discontinued. There were, in the end, a total of 22 distinct software testing courses. Four more courses were omitted from consideration as the authors are involved, in varying degrees, with the courses. Eighteen invitation emails were sent out thus forming a consensus, as invitations were sent to the entire target population.

B. Pilot Study

Pilot studies can help to ensure that the questions asked are clear, comprehensive and aligned with the objectives. With online surveys, syntax issues such as branching options can also be evaluated. While conducting pilot studies does not automatically guarantee successful surveys, doing so could increase the likelihood of a successfully completed survey [30].

With that in mind, four academic staff members at a Swedish university were invited to take part in a pilot study for this survey. The experience levels ranged from a PhD student to two postdoc researchers and one lecturer. Having been involved in the teaching of software testing at different stages of their careers, these academics were representative of the sample population. Areas of concern raised during the pilot study were reviewed and addressed by the authors before the questionnaire being sent to the target population.

[1] https://www.uhr.se/en/start/

C. Description of the Questionnaire

A lengthy survey instrument has been determined to be one reason for respondent fatigue [23]. Accordingly, the questionnaire was limited in the number of questions, ensuring only pertinent questions were asked. The questionnaire was made up of five parts. The first section consisted of only one question relating to consent. A negative response would have directed the respondent to the end of the questionnaire, signifying that no participation consent was given. An answer in the affirmative would have directed the respondent to Section 2 of the questionnaire. The questions are repeated in Table I, albeit paraphrased for brevity.

The survey was intentionally kept simple in its design. In addition to the first question on consent, there were just two other conditional questions which became relevant only if a specific value from a preceding question was selected. For example, in Section III, the question seeking the names of the test tools used would only have been shown to the respondent if the previous question had been answered in the affirmative. Thus, the questionnaire was fairly easy to complete. Except for an outlier respondent who took just under three hours to complete the survey, the average completion time of the survey was about 10 minutes. This was similar to the average times in the pilot study.

TABLE I
SURVEY QUESTIONS

Section	Question	Input Type
I	Consent to survey?	Single selection
II	Responsible academic department:	Single selection
	Applicable teacher role(s) used:	Multiple selection
	Instructional materials used:	Multiple selection
	Textbooks used:	Free text
	Student assessment types:	Multiple selection
	Course delivery mode:	Single selection
III	Knowledge base of course contents:	Single selection
	Basis of test case:	Single selection
	Use of test tools in class?	Single selection
	Test tool(s) used:	Free text
	Test domain(s) taught:	Multiple selection
IV	Test design techniques covered:	Scale
	Additional test techniques taught:	Free text
	Test levels coverage:	Scale
	SDLC coverage:	Scale
	Test types coverage:	Scale
	Other test-related topics covered:	Multiple selection
	Test automation coverage:	Multiple selection
V	Potential new areas of testing:	Free text

D. Response Rate

Response rates of survey-based studies have been used to judge their quality and reliability [10]. What constitutes an acceptable response rate, however, is yet to be universally agreed upon [28]. It has been suggested that there is no direct correlation between the validity of surveys and response rates and that surveys with relatively low rates can be able to achieve reasonable results. Nevertheless, while surveys with relatively low rates can achieve acceptable results, every effort should be made to increase the response rate [9, 31].

From the 18 invitations sent, the survey was initially completed seven times. Although the questionnaires were anonymous, three respondents emailed to confirm the completion of the questionnaire. Given the initial response rate of 39%, and in an attempt to increase the response rate, a second round of 15 emails was sent as a reminder for those who may not yet have completed the survey. The reminder increased the number of responses marginally by one. As such, with a total of eight responses received, the overall response rate was 44%.

IV. RESULTS

Descriptive statistics are used to describe and organise the characteristics of data garnered on software testing courses in Sweden. The following subsections will contain summaries and descriptions of a select number of questions from three sections in the survey:

1) Course details and structure
2) About the software testing course
3) Course content

A. Course Details and Structure

The questions in this section of the questionnaire were asked specifically to elicit information that will help in answering our RQ1. There are a total of six questions in this section (see Table I). Some of the questions are discussed in the following sub-sections.

1) Course Custodian: The terms "Computer Science", "Software Engineering" and "Information Technology" are sometimes used interchangeably. The corresponding departments, however, are different with specific areas of focus albeit with areas of overlap [6].

Figure 1 depicts which department is primarily responsible for offering the software testing courses. Half of the courses evaluated are offered in the Department of Software Engineering. Three of the courses were offered in Computer Science departments, and only one was offered in the Department of Information Technology.

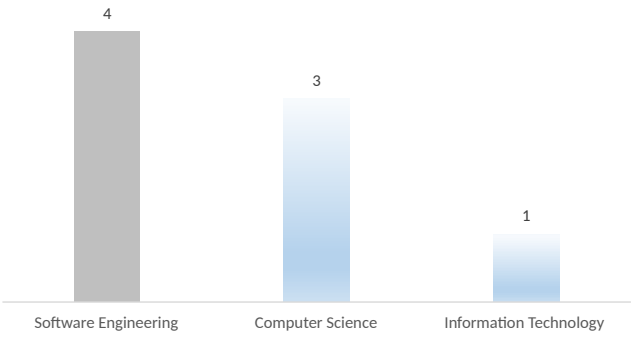

Fig. 1. Departments responsible for software testing courses

2) Teacher Roles: In academic institutions, tasks, duties and responsibilities associated with courses differ from role to role. For example, decision-making activities (e.g. determining course content or the choice of recommended textbooks) are more often associated with lecturers. Typically, teaching assistants' roles may span from undertaking non-instructional tasks to delivering instructions that are complementary and supplemental to those of the lecturer and, often, work more closely with students than lecturers [26, 32].

Figure 2 outlines the teacher roles for teaching software testing. All of the courses have lecturers assigned to the courses, with only one course utilising a teaching assistant. With regard to the use of industry guest lecturers, three of the courses invite external expertise. There is no evidence of guest lecturers from other parts of the university or from other universities.

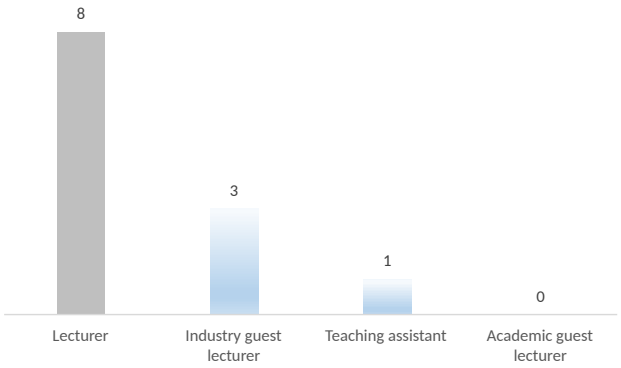

Fig. 2. Teacher roles involved in teaching testing courses

3) Instructional Materials and Course Books: Additional information was sought in the survey about instructional materials used, how students registered for the course are assessed and how the testing course is delivered. The responses show that the courses are well-grounded theoretically, with seven of the eight courses utilising textbooks and six of eight courses recommending academic research articles. Five textbooks were used in the courses, with the book by Ammann and Offutt [1] being the most commonly recommended book in three instances. One of the courses, however, neither used textbooks nor articles using only videos and lecture slides to convey information.

4) Course Delivery Modes and Student Assessment Types: Concerning course attendance, four courses require students to attend lectures for face-to-face interactions, three of the courses are offered as hybrid courses, and one course is offered entirely online. Student assessments, which help lecturers gauge how well students understand course contents, encompass the entire range of options offered in the survey, albeit in varying numbers. The least-used forms of assessments are lab sessions and individual projects, at two a-piece. Conversely, the most used form of assessment is assignments, which are used in seven of eight courses.

The first set of questions in the survey were asked to gather information needed to assist in answering the first research question regarding the structure of software testing courses in Swedish universities. In summary, half of the courses are located in the Software Engineering department, but a good number are in the Computer Science department. Few industrial guest lecturers or research assistants are utilised in teaching courses, one-half of the courses require students to attend physical classes and, the most-commonly assessment type is assignments.

B. About the Software Testing Course

In this section of the questionnaire, questions were asked to gain an understanding of the key characteristics of the courses. These questions were asked to aid in answering the second research question. Responses to questions related to what forms the basis of the course contents, what derived test cases are based on, which test tool types are utilised and which test domains are focused upon in the courses are provided in the following sub-sections.

1) Underlying Knowledge Base: In the development of two of the courses, the contents were primarily derived from textbooks. One of the courses is formed, for the most part, from the syllabus of a software testing certification body, while yet another is based on an external body of knowledge (such as the Software Engineering Body of Knowledge, or SWEBOK). For two responses provided via the "Other" option, one response highlights the course content's evolution based on the instructor's experience and industry feedback. Two of the responses, however, indicate that the courses do not adhere to any formal standards or external body of knowledge. The responses are shown in Figure 3,

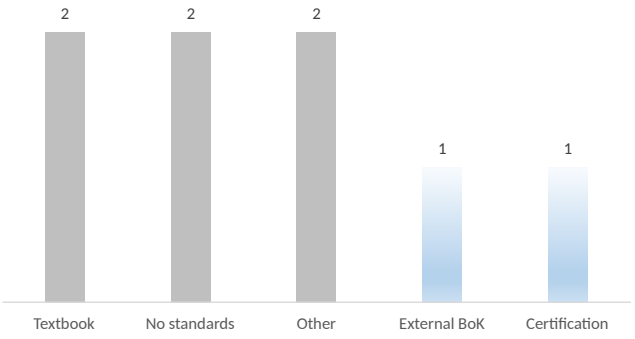

Fig. 3. Source of course content

2) Test Case Basis: A test basis, also known as the system under test (SUT), refers to the software being tested. These can differ in size from as little as a module, consisting only of a few lines of code, to the entire application. During software testing, test cases are run against the test basis to unearth any defects [25].

As Figure 4 shows, two primarily use real-world projects to drive the testing effort of the eight courses surveyed. One

other course uses a university project developed in a separate, concurrent course to drive the testing effort. Five responses provided additional responses via the "Other" option *viz.*: (i) Lab exercises (ii) Some open-source, some toy examples (iii) Unit tests for small-size software (iv) Test cases for real-world web pages as well as a toy example (v) Designed labs in python.

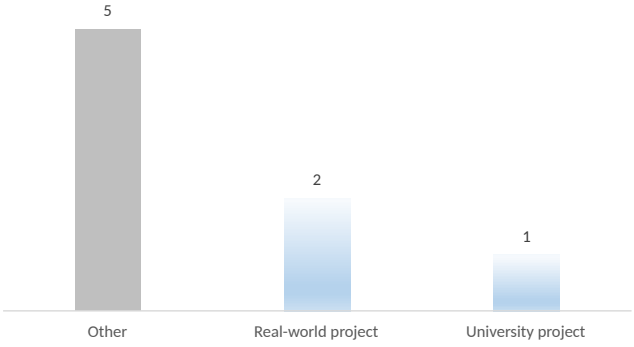

Fig. 4. Test case basis

3) Test Tools Used: All the courses use test tools, which are documented in Table II and listed in decreasing order of use. The number in parenthesis signifies how many courses use the specific testing tool; for example, JUnit is utilised in four of the eight courses. Tools without parenthesis signify that the tool is used in only one course.

TABLE II
TEST TOOLS USED. THE NUMBER IN PARENTHESIS IS NUMBER OF COURSES USING A TOOL.

Test area	**Test tools**
Unit testing	xUnit/JUnit (6); Evosuite; PyTest
Website testing	Selenium (2)
Mutation testing	PIT (2); muJava
API testing	Postman
GUI testing	Eyeautomate
UAT testing	Scout
Property-based testing	Hypothesis; jqwik
Static analysis	SonarQube

4) Test Domain: Given the prevalence of mobile phones, to our little surprise, only one of the courses taught testing for mobile technologies. Conversely, and just as surprising, most courses were based on desktop application testing. In one of the responses ("other"), the respondent stated that the focus of the course was unit testing; thus, no specific domain was covered in the course.

The questions in this section were asked to gain the understanding needed to answer the second research question of key "characteristics" typical of software testing courses in Swedish universities. To summarise, with a vast majority of the courses, standardised material is used to determine what core areas are taught. Nevertheless, most of the courses did not use real-world projects to underscore the teaching of testing aspects. All the courses used test tools, demonstrating a practical approach to teaching the course contents.

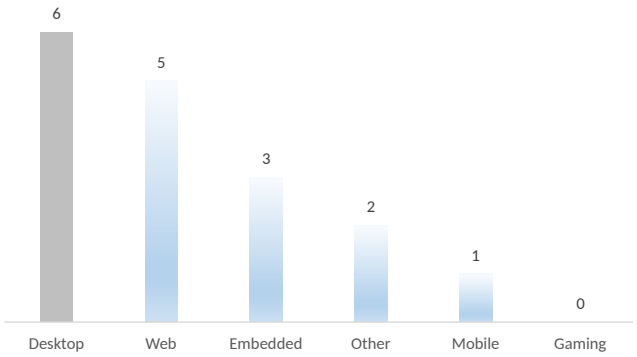

Fig. 5. Test domains taught

C. Software Testing Course Content

In this section of the questionnaire, questions sought to ascertain the depth and breadth of the content of the testing courses. Specifically, answers were asked regarding test design techniques, test levels, test types, development methodologies and automation. In doing so, the third research question will be answered.

1) Test Design Techniques: Myriad test design techniques have been identified in the literature. These techniques are utilised to provide structure to the testing exercise and, ideally, help reduce the number of test cases while improving chances of finding defects [1]. Figures 6 and 7 depict the most-taught and least-taught test design techniques, respectively, from the survey results.

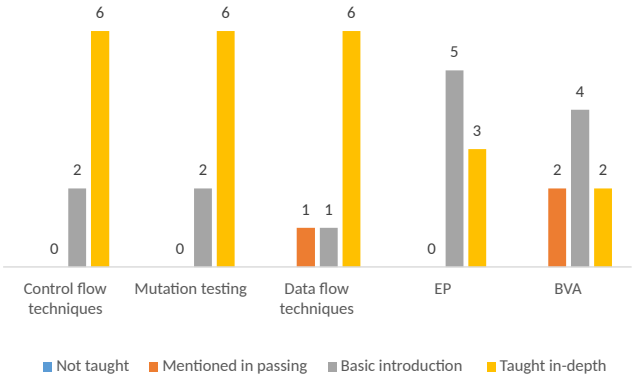

Fig. 6. Most-frequently taught test design techniques

Two techniques shared the top spot as being the most-taught techniques. These are control flow techniques and mutation testing; six courses cover both techniques in-depth and two courses provide a basic introduction to the techniques. Data flow techniques are the third-most common technique taught, with six courses covering the technique in detail. Still, one course provides a basic introduction to the technique, and the

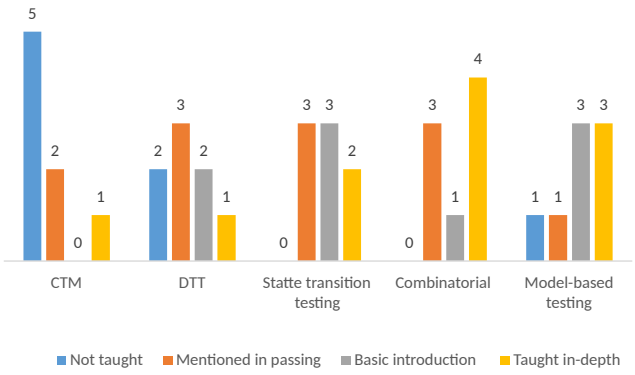

Fig. 7. Least-frequently taught test design techniques

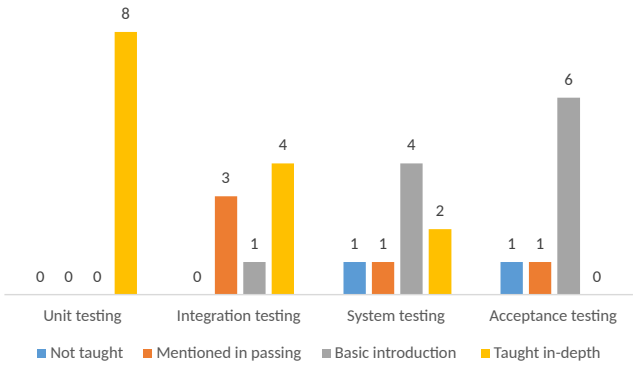

Fig. 8. Test levels taught

technique is mentioned in passing only in yet another course. Equivalence partitioning (EP) and boundary value analysis (BVA) round up the top-five test design techniques.

The least-taught test design technique is the classification tree method (CTM), which is omitted from five of the courses, and in a further two courses, the technique is mentioned only in passing. The second least-taught technique is decision table testing (DTT), in which no mention of the technique is made in two courses, and the technique is mentioned in passing in three courses. These results are dissimilar to the findings by [21], who found in their study that boundary value analysis and equivalence partitioning were the most frequently taught test design techniques. For one of the courses, further information was provided, indicating that exploratory testing was an additional test design technique covered in the course.

2) Test Levels: Test levels relate to corresponding software development activities and artefacts. Unit and integration tests are derived from software code and subsystem design, respectively. The former assesses code units, and the latter assesses the interfaces between the units of code. System tests, derived from architectural designs, are run against the system once it has been assembled together as a whole. Acceptance tests, formed from documented user requirements, are run to determine if the completed software meets users' needs. Ammann and Offutt [1] argue that each level needs to be tested as some faults can only be found at different levels.

As shown in Figure 8, unit testing is the most-taught test level, indicating that the test level is covered in-depth in all the courses. In-depth coverage of integration testing is half that of unit testing. The number of courses in which this test level is covered in-depth is further halved to just two courses at the system level. The least-covered level is the acceptance test level, with no course indicating that the topic was covered in-depth. The vast majority of the courses, six out of the courses, provide only a basic introduction to the topic.

3) Software Development Methodology: The method by which software is developed systematically, also known as software development life-cycles (SDLCs), guides a series of phases and activities for all members of the development team [22]. These methodologies fall within two broad groups *viz.* sequential methodologies (e.g. the waterfall, V- and W-models) and iterative methodologies (e.g. Scrum, Kanban, Lean, eXtreme Programming and DevOps). Regardless of which category that an SDLC may belong, software testing is a vital phase in all development life cycles.

Depicted in Figure 9 is the extent to which sequential methodologies are covered in the testing courses. The two sequential methodologies are covered in-depth in only three of the courses. In three other courses, these methodologies are not taught at all. Similarly, Figure 10 depicts to what extent iterative methodologies are covered. The most taught iterative methodology is Scrum, which covers the topic in-depth or is used exclusively to demonstrate testing in three courses. Conversely, the least-covered iterative methodology is Kanban, with four courses not covering the topic.

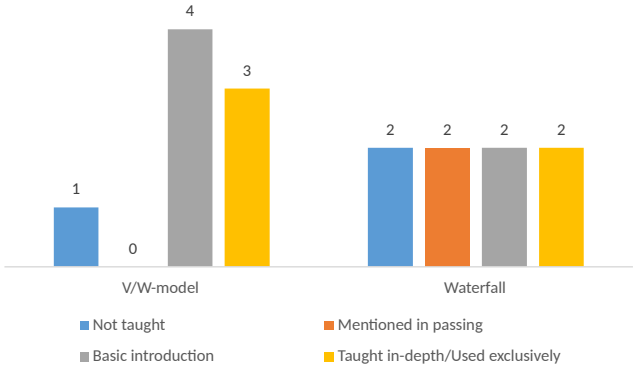

Fig. 9. Sequential methodologies

4) Test Types: Software should be tested to ensure that required functionalities are met, that is, the software works as intended. Additionally, attributes, or the behaviour that constrains how well the software works as it ought to, should

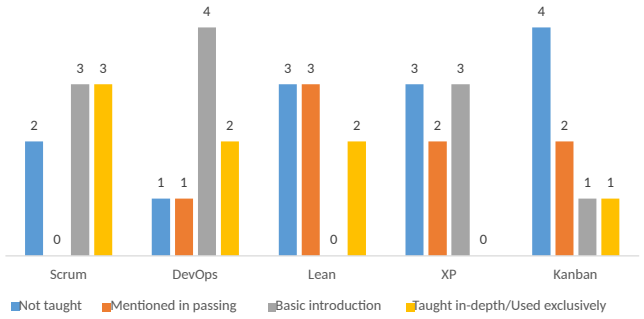

Fig. 10. Iterative methodologies

be tested. With the latter, tests should be carried out to ascertain how well the software responds to normal to excessive demands (performance, load and stress testing) and how easily users can use the software, especially those with imitations (usability and accessibility testing). Other attributes that should be tested include how easy the software will be to maintain once it is entirely in use (maintainability), how secure the software will be from malicious intent (security) and how well the software will work for extended periods without failures and how quickly recovery would occur after a failure (reliability).

Question 17 of the survey sought to determine how much these test types are covered in the courses. The five most-covered test types are shown in Figure 11. Functional testing is covered in almost all the courses. Change-based testing, load testing and usability testing are all covered in-depth in two courses, with performance testing being taught in-depth in only one course. Load and usability testing are not covered in detail in three of the courses. With performance testing, the topic is not taught in two courses. The five least-covered topics are shown in Figure 12. Stress and security testing are not taught in three courses, but both are taught in-depth in one course each.

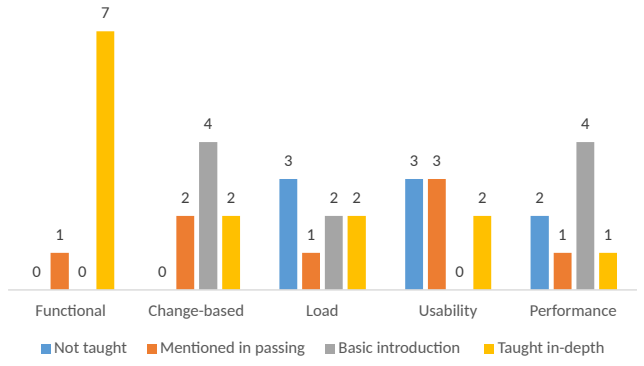

Fig. 11. Test types (Most taught)

5) Test Related Topics: Testing is far more involved than only looking for defects. Additional related activities could assist in improving the overall quality of the software. Examples include reporting the outcome of the testing process and static testing techniques, such as reviews and inspections, that have been described as highly effective techniques that contribute to improved software quality. This is because the detection and removal of defects could occur well before even the software code is written [5, 27]. Nevertheless, half of the courses do not include this quality assurance perspective in this survey. The most-taught, test-related topic, however, is using test metrics. Static testing using tools and test planning activities came in as close seconds. Besides simulation, which is not covered in any of the courses, risk-based testing (RBT) is the least-covered topic.

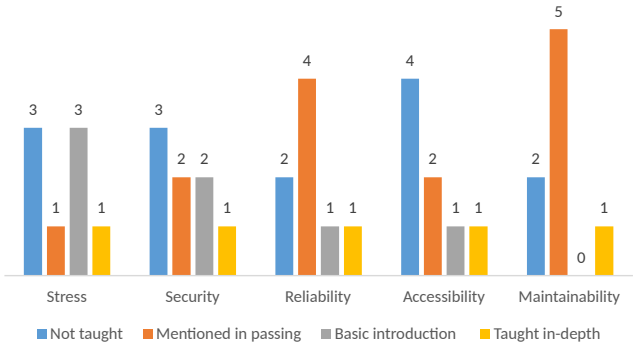

Fig. 12. Test types (Least taught)

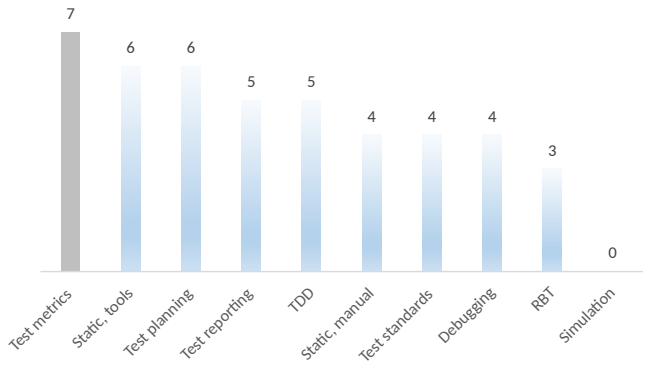

Fig. 13. Test-related Topics

D. Additional Insights

1) Test Domain vs Test Type: Different domains are covered in the courses surveyed (Section IV-B4). In this section, a test domain, specifically the web domain, is juxtaposed against appropriate test types (Section IV-C4). Although web testing is not the most common domain taught in the courses detailed in this survey, it is the second-most common domain and, by nature of its characteristics, does offer more opportunities for demonstrating more test types than the testing of desktop applications.

TABLE III
WEB DOMAIN TESTING AND SYLLABI COVERAGE OF TEST TYPES

	Performance	**Load**	**Stress**	**Usability**	**Accessibility**
C3	Basic	Basic	Basic	In passing	In passing
C5	Not taught	Not taught	Not taught	Not taught	Not taught
C6	In-depth	In-depth	In-depth	In-depth	Basic
C7	Basic	Basic	Basic	In passing	In passing
C8	In passing	Not taught	Not taught	In passing	Not taught

Web testing is the testing of web applications in an attempt to detect defects before the website being made available to users. In addition to ensuring that the website would work as required (functional testing), and due to the nature of the domain, there are typically other test types that are done. Being a user-facing application, examples include testing the usability and accessibility of the website. Additionally, as multiple users use websites simultaneously, performance, load and stress testing ideally should occur.

There are five of the courses for which there are indications that the web domain was covered in the respective syllabi (see Figure 5, page 5). Coverage of applicable test types, reproduced in Table III, run the gamut. For example, Course C5 does not include teaching any of the test types typically undertaken with testing of web applications. At the same time, Course C6, save for accessibility testing, covered all the topics in-depth.

It must also be pointed out that while usability testing is taught in-depth in Course C3, accessibility testing, often thought to occur hand-in-hand with usability testing, is not given as much prominence. Accessibility testing should be done to ensure that people with disabilities can use software, with or without the aid of assistive technologies [13].

V. INSIGHTS

Responses to the survey were collected from lecturers teaching software testing at eight Swedish universities. Most of the courses are situated in the Software Engineering department. While most of the courses are based on desktop applications as systems under test, a relatively good mix of domains was used to demonstrate different testing topics.

The survey results offer some possible insights. Specifically, issues relating to teacher roles, sources of course content, system under test (SUT), test domains, test levels and, lastly, test techniques taught will be discussed briefly in the following paragraphs.

a) Need of guest lectures from industry and academia: The first to be discussed is the issue of the teacher roles utilised in the teaching of the courses. Fewer than half of the courses make use of guest lecturers. The use of guest lecturers, particularly those outside of academia, may trigger a sense of excitement in students. Guest lecturers are also thought to give credence to a subject area or demonstrate possible industry opportunities [14, 35]. As such, for a discipline that students have described as "boring" [34, p. 1498], "bland, unexciting and insignificant" [15, p. 1795] or, "monotonous, not exciting, repetitive" [7, p. 1], practitioners, as guest lecturers, may, for example, provide real-world examples where software testing made significant contributions. Similarly, guest lectures from researchers on timely testing topics such as testing for security and testing for intelligent systems with built-in artificial intelligence capabilities, will also help address the breadth of testing topics within a limited time of a course offering.

b) Integrate contents from books, standards, certifications and industry: Regarding the source of course content, four courses rely on the use of textbooks, recognised standards or certification bodies, suggesting a structured approach to developing the course contents. Also important to note is the inclusion of industry feedback in one of the courses, which suggests a dynamic and responsive approach to teaching. There are, however, two courses that do not adhere to any formal standards or external body of knowledge. This could lead to variability in the breadth and depth of the topics covered. In addition to course books and inclusion of industrial feedback, certification bodies like ISTQB (International Software Testing Qualifications Board) and standards like ISO/IEC/IEEE 29119 have gathered valuable body of knowledge from experts that promises to improve current testing course content.

c) Provide realistic SUT: During software testing, test cases are run against the SUT to detect defects [25]. Two courses use "toy examples" to demonstrate testing concepts, and three other courses demonstrate software testing with lab exercises. Including real-world projects from open-source repositories [8] or companies, as done in two of the courses, could provide valuable, practical testing experience. The more realistic a test base is, the more likely students will be engaged with the concepts taught. Realistic SUTs may also help develop a greater appreciation of the utility of software testing as a profession and possibly increase student preparedness for working as a tester in industry [16].

d) Expand test domains: Given the prevalence of mobile phones, only one of the courses taught testing for mobile technologies. Conversely, and just as surprising, the majority of the courses were based on desktop application testing. With one of the responses ("other"), the respondent stated that the focus of the course was unit testing, thus no specific domain was covered in the course. Switching among test domains, for example, in different assignments, should be given serious thought to diversify students' learning outcomes.

e) Provide education in test levels and their role in SDLC: In all the courses, unit-level testing is taught in-depth. In sharp contrast, the number of courses that teach

acceptance level in detail drops to zero. Different members of the development team are likely to undertake testing at different test levels, depending upon the different stages of SDLC. Also, the defects that can be found will differ according to the test level. It will be prudent to provide adequate skills so that each level can be handled with skilled software engineering team members.

f) Expand education of test design techniques: Related to the previous discussion on integrating course contents from diverse sources, the survey results show that there is room to expand the coverage of test design techniques in existing courses. It is suggested that a coverage of test design techniques based on an analysis of the content of relevant books, standards, certification bodies, and industrial practices.

VI. Threats to Validity

One potential threat to internal validity, often associated with using questionnaires, is a misinterpretation of questions by respondents. Linguistically, this threat was greatly minimised as respondents were concentrated around the same geographical region and language. Nonetheless, conducting a pilot study before the questionnaire was sent out reduced the possibility of this risk. Additionally, construct validity may be threatened by the broad specificity of the survey questions, which could overlook specific topics in course content. External validity is another concern, as the findings are based on a limited sample of courses from Swedish universities, which may differ from global software testing education. In addition, sampling bias poses a challenge, as the study achieved a response rate of 44%. Efforts were made to engage various university respondents to capture a broader perspective on software testing education.

VII. Conclusions

This paper investigates the topic of academic software testing courses in the context of their structural, underlying, and core aspects. The results draw attention to the fact that much of the course content is focused on programmer-aligned testing. This is shown not only by the high level of in-depth coverage of unit testing but also by the category of typical test tools used. The survey results offer insight into issues relating to teacher roles, sources of course content,system under test (SUT), test domains, test levels, and test techniques taught.

Future work includes widening the research globally and comparing results from other countries. Further consideration should also be given to additional data such as student demographics, the level of study and student numbers per registration cycle. Additionally, the results from the survey suggest other areas of research. As an example, Question 3 of the survey shows that there are relatively few courses that make use of industry guest lecturers. It would be of interest to empirically determine if their inclusion in the teaching of software testing could change students' perspective about the subject.

Acknowledgment

The support of Mälardalen University (MDU), through the Promote Lifelong Learning (FLL) grant and ModTest project, is acknowledged. FLL is an MDU university-wide project aimed at developing and strengthening opportunities for MDU to offer education to those looking for further training, upskilling, and re-skilling. Eduard Enoiu is also supported by the SmartDelta project (funded by Vinnova) and by the Software Center project.

The authors thank the lecturers of the software testing courses who took time out of their busy schedules to complete the survey. There is great appreciation also for the reviewers of the survey during the pilot study for their invaluable comments and constructive critique.

References

[1] P. Ammann and J. Offutt. *Introduction to Software Testing*. Cambridge University Press, New York, 2013.

[2] B. Ardic and A. Zaidman. Hey teachers, teach those kids some software testing. In *2023 IEEE/ACM 5th International Workshop on Software Engineering Education for the Next Generation (SEENG)*, pages 9–16, 2023.

[3] A. A. Barrett, E. Paul Enoiu, and W. Afzal. On the current state of academic software testing education in Sweden. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 397–404, 2023.

[4] B. Cerón, A. Chicaiza, J. Lamiño, and R. P. R. Ch. Software testing in Ecuadorian university education: A debt to the software industry. In Á. Rocha, H. Adeli, G. Dzemyda, F. Moreira, and A. Poniszewska-Marańda, editors, *Good Practices and New Perspectives in Information Systems and Technologies*, pages 80–90, 2024.

[5] M. Ciolkowski, O. Laitenberger, S. Vegas, and S. Biffl. Practical experiences in the design and conduct of surveys in empirical Software Engineering. In R. Conradi and A. I. Wang, editors, *Empirical Methods and Studies in Software Engineering: Experiences from ESERNET*, pages 104–128. 2003.

[6] S. E. Conry. Software Engineering, Computer Engineering, Computer Science: Sibling disciplines with diverse cultures. In *2011 ASEE Annual Conference & Exposition*, pages 22–1308, 2011.

[7] A. Deak, T. Stålhane, and G. Sindre. Challenges and strategies for motivating software testing personnel. *Information and software Technology*, 73:1–15, 2016.

[8] L. Deng, J. Dehlinger, and S. Chakraborty. Teaching software testing with free and open source software. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2020.

[9] K. Fosnacht, S. Sarraf, E. Howe, and L. K. Peck. How important are high response rates for college surveys? *The Review of Higher Education*, 40(2):245–265, 2017.

[10] B. Holtom, Y. Baruch, H. Aguinis, and G. A. Ballinger. Survey response rates: Trends and a validity assessment framework. *Human Relations*, 75(8):1560–1584, 2022.

[11] F. Kazemian and T. Howles. A software testing course for Computer Science majors. *SIGCSE Bull.*, 37(4):50–53, 2005.

[12] B. A. Kitchenham and S. L. Pfleeger. Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer, 2008.

[13] P. Kotzé, M. Eloff, A. Adesina-Ojo, and J. Eloff. Accessible computer interaction for people with disabilities – The case of quadriplegics. In *ICEIS 2004-Proceedings of the Sixth International Conference on Enterprise Information Systems*, 2004.

[14] B. R. Krogstie and J. Krogstie. Guest lectures in IT education - Recommendations based on an empirical study. 2018.

[15] D. E. Krutz and M. Lutz. Bug of the day: Reinforcing the importance of testing. In *2013 IEEE Frontiers in Education Conference (FIE)*, pages 1795–1799, 2013.

[16] D. E. Krutz, S. A. Malachowsky, and T. Reichlmayr. Using a real world project in a software testing course. In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 49–54, 2014.

[17] O. A. L. Lemos, F. C. Ferrari, F. F. Silveira, and A. Garcia. Experience report: Can software testing education lead to more reliable code? In *Proceedings of the 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, ISSRE '15, page 359–369, 2015.

[18] J. Linåker, S. M. Sulaman, R. M. de Mello, and M. Höst. *Guidelines for conducting surveys in Software Engineering*. Department of Computer Science, Lund University, Sweden, 2015.

[19] W. Marrero and A. Settle. Testing first: Emphasizing testing in early programming courses. In *Annual Conference on Innovation and Technology in Computer Science Education*, 2005.

[20] T. Mårtensson and K. Sandahl. The testing hopscotch model – Six complementary profiles replacing the perfect all-round tester. In R. Kadgien, A. Jedlitschka, A. Janes, V. Lenarduzzi, and X. Li, editors, *Product-Focused Software Process Improvement*, pages 495–510, 2024.

[21] S. M. Melo, V. X. S. Moreira, L. N. Paschoal, and S. R. S. Souza. Testing education: A survey on a global scale. In *Proceedings of the XXXIV Brazilian Symposium on Software Engineering*, SBES '20, page 554–563, 2020.

[22] A. Mishra and D. Dubey. A comparative study of different software development life cycle models in different scenarios. *International Journal of Advance research in computer science and management studies*, 1(5), 2013.

[23] S. P. Ng, T. Murnane, K. Reed, D. Grant, and T. Y. Chen. A preliminary survey on software testing practices in Australia. In *2004 Australian Software Engineering Conference. Proceedings.*, pages 116–125. IEEE, 2004.

[24] M. C. Paulk, B. Curtis, M. B. Chrissis, and C. V. Weber. Capability Maturity Model for software, version 1.1. *IEEE Software*, 10(4):18–27, 1993.

[25] F. Raab. System under test. In S. Sakr and A. Zomaya, editors, *Encyclopedia of Big Data Technologies*, pages 1–3. Springer International Publishing, Cham, 2018.

[26] U. Sharma and S. J. Salend. Teaching Assistants in Inclusive Classrooms: A Systematic Analysis of the International Research. *Australian Journal of Teacher Education*, 41(8):118–134, 2016.

[27] T. Shepard, M. Lamb, and D. Kelly. More testing should be taught. *Communications of the ACM*, 44(6):103–108, 2001. ISSN 0001-0782.

[28] S. A. Sivo, C. Saunders, Q. Chang, and J. J. Jiang. How low should you go? Low response rates and the validity of inference in IS questionnaire research. *Journal of the association for information systems*, 7(6):351–414, 2006.

[29] P. Tramontana, B. Marín, A. C. R. Paiva, A. Mendes, T. E. J. Vos, D. Amalfitano, F. Cammaerts, M. Snoeck, and A. R. Fasolino. State of the practice in software testing teaching in tour European countries. In *17th IEEE International Conference on Software Testing, Verification and Validation (ICST) 2024*, 2024.

[30] E. Van Teijlingen and V. Hundley. The importance of pilot studies. *Social Research Update*, 35:1–4, 2001.

[31] W. Wiersma. The validity of surveys: Online and offline. *Oxford Internet Institute*, 18(3):321–340, 2013.

[32] M. Wilmore. What's in a name? Reflections on working as a 'teaching assistant' at University College London and as an 'associate lecturer' at The Open University. *Anthropology Matters*, 5(1):–, 2003.

[33] W. E. Wong. Improving the state of undergraduate software testing education. In *2012 ASEE Annual Conference & Exposition*, San Antonio, Texas, June 2012. ASEE Conferences.

[34] T. Zivkovic, D. Draskovic, and B. Nikolic. Learning environments in software testing education: An overview. *Computer Applications in Engineering Education*, 31(6): 1497–1521, 2023.

[35] P. Zou, W. Sun, S. G. Hallowell, Y. Luo, C. Lee, and L. Ge. Use of guest speakers in nursing education: An integrative review of multidisciplinary literature. *Advances in medical education and practice*, pages 175–189, 2019.